Tutorial on High Performance Multithreaded Packet Processing

Juha-Matti Tilli, Nokia Bell Labs

This tutorial considers high performance multithreaded packet processing using software running on Linux computers. Firstly, we consider the recent and oncoming developments in microprocessors and mobile packet data networks to see that packet processing at desired rates has indeed become very hard. Then we take a look at multithreading to see where it can help and where it cannot help. Multithreading today is achieved by packet hashing engines in network interface cards (NICs), and we discuss the various ways packets can be hashed to separate cores. Other functionalities of NICs such as interrupt mitigation are also discussed.

To actually achieve the multithreaded packet processing capabilities, support from the programming environment is needed. Linux kernel has support for several features intended to increase performance: multiple message receive/send, packet mmap for raw sockets and fanouts. Unfortunately, the Linux kernel transmission control protocol / Internet protocol (TCP/IP) stack is too slow, so if highest possible performance is needed, it needs to be bypassed. We look at several recent solutions for bypassing the kernel stack, including data plane development kit (DPDK), netmap and the wrapper around various technologies, OpenDataPlane (ODP). The commercial PF_RING is also briefly mentioned. It is observed that if energy efficiency is desired in addition to high performance, targeting netmap directly is the most sensible thing to do.

Because most traffic in today’s networks is TCP, we discuss also TCP offloads. The offloads started to appear first in the form of checksum offloads both for receive and send side. Then, send segmentation offload and receive reassembly offload appeared. It is discussed why receive reassembly offload is not fully transparent and should be thus disabled in some cases. The merits and drawbacks of using a large maximum transmission unit (MTU) are also discussed, and it is seen that with modern NICs, large MTU in local networks is not as beneficial as it used to be.

Several different architectures for high performance multithreaded packet processing are introduced. The architectures have different characteristics and different uses may have a different optimal architecture, so there is no single best architecture applicable for all cases. It is explained how binding threads to CPU cores can improve performance dozens of percent, and it is also explained how crossing non-uniform memory architecture (NUMA) boundaries can reduce performance with increasing thread count. Potential deadlocks in certain packet processing architectures are also identified.

Tunneling protocols are discussed and it is seen what kinds of challenges tunneling brings to optimal packet hashing. Solutions to the challenges are presented for protocols running on top of UDP and directly on top of IP.

To support timers, we discuss how thread local timers are best implemented. It is seen that the optimal data structure for timers is a heap. However, if there are a large number of timers for a huge number of
state table entries and sub-second accuracy is not needed, it is seen how timers can be batched for smaller count of timers in the heap in a way that doesn’t cause latency to skyrocket.

Inter-thread communication queues are shortly discussed. It is seen how burst mode can improve performance many-fold. Queue cache allows easily transmitting packets in bursts, and timed burst mode dequeue operation allows easy interoperation with timers.

It is found that for linked list and separately chained hash tables, it makes sense to embed the list nodes inside the data structures and use Linux kernel style CONTAINER_OF macro to obtain structure itself from list node.

Then hash tables are discussed. It is explained why today it may make more sense to have power of 2 hash table sizes instead of prime hash table sizes. Fast division by invariant integers by multiplication is also briefly mentioned. The details of implementing a good hash table in C language are discussed, and it is found that Linux kernel hash table is almost optimal with dually-linked middle nodes yet singly-linked heads. Hash functions are also discussed, and it is recommended that SipHash is used where hash collision attacks are a possibility and MurMurHash3 where hash collision attacks are not a possibility.

Address resolution protocol (ARP) and its IP version 6 (IPv6) equivalent neighbor discovery protocol (NDP) is something that each layer 3 network element must support. It is discussed how ARP can be implemented in a lock-free thread-local way in certain packet processing architectures.

Implementing control plane on top of the user plane for certain packet processing architectures is also discussed. It is recommended that blocking queues support two priorities with control plane operations having higher priority than user plane data packets. The control plane thread can wait for user plane thread to process the messages synchronously. If the user plane needs ability to trigger control plane operations, a split input/output threaded architecture is also discussed.

The support of various 40 gigabit NICs are discussed. Mellanox cards are widely used, but unfortunately they have only DPDK support. Chelsio on FreeBSD supports netmap but on Linux not. So, the only choice for high netmap performance on Linux is Intel’s 40 gigabit cards.

Running multiple netmap applications on the same computer is also discussed. One possibility is to implement a tap device forwarding application that allows multiple ports per virtual switch. However, netmap’s VALE offers much improved performance over this approach. Also, Linux kernel has veth driver but it is extremely slow and supports only two ports.

At the end, we look at actual code that implements interfacing with netmap. It is seen that the application programming interface (API) of netmap is really simple and easy to learn. We also look at how the worker thread functionality is implemented.

We also run some benchmarks on a real netmap application implementing one of the mentioned packet processing architectures.
Juha-Matti Tilli obtained his B.Sc. in electrical engineering in 2014 and his M.Sc. in 2015 from Aalto University. His major was micro- and nanotechnology. However, most of his professional career has been working with Internet-based services and communications networks and equipment. His research interests include network test traffic generation, performance measurement and high-speed packet processing. He has started his Ph.D. studies in communications engineering in late 2015. He currently works at Nokia Bell Labs.